Linked List Implementation III

26 min

Nice! Now we have a bunch of helpful `LinkedList` methods under our belt.

The final use case we mentioned was the ability to remove an arbitrary node with a particular value. This is slightly more complex, since a couple of special cases need to be handled.

Consider the following list:

```
a -> b -> c
```

If node `b` is removed from the list, the new list should be:

```
a -> c
```

We need to update the link within the `a` node to match what `b` was pointing to prior to removing it from the linked list.

Lucky for us, in Python, nodes which are not referenced will be removed for us automatically. If we take care of the references, `b` will be "removed" for us in a process called Garbage Collection.

For the purposes of this lesson, we'll create a function that removes the *first* node that contains a particular value. However, you could also build this function to remove nodes by index or remove all nodes that contain a particular value.

Instructions

1.

At the bottom of **script.py**, add a `.remove_node()` method to `LinkedList`. It should take `value_to_remove` as a parameter. We'll be looking for a node with this value to remove.

In the body of `.remove_node()`, set a new variable `current_node` equal to the `head_node` of the list.

We'll use `current_node` to keep track of the node we are currently looking at as we traverse the list.

Hint

You can retrieve `head_node` using the `.get_head_node()` method you built out!

2.

Still inside the method body, use an `if` statement to check whether the list's `head_node` has a value that is the same as `value_to_remove`.

If it does, we've found the node we're looking for and we need to adjust the list's pointer to `head_node`.

Inside the `if` clause, set `self.head_node` equal to the second node in the linked list.
Hint
You can use `Node`'s `get_value()` method to retrieve `head_node`'s value.

Remember that the list's second node is the current `head_node` 's `next_node`.

3.

Add an `else` clause. Within the `else` clause:

- Traverse the list until `current_node.get_next_node().get_value()` is the `value_to_remove`.

(Just like with `stringify_list` you can traverse the list using a `while` loop that checks whether `current_node` exists.)

- When `value_to_remove` is found, adjust the links in the list so that `current_node` is linked to `next_node.get_next_node()`.
- After you remove the node with a value of `value_to_remove`, make sure to set `current_node` to `None` so that you exit the loop.

Hint
We recommend that you create a variable `current_next_node` to keep track of `current_node`'s `next_node`.

Consider `a -> b -> c` again:

If `a` is `current_node`, `current_node.get_next_node()` is `b`.

If `b.get_value() == value_to_remove` is `True`, you want to set `a`'s `next_node` property to `c`.

script.py

```
# We'll be using our Node class
class Node:
  def __init__(self, value, next_node=None):
```

```
    self.value = value
    self.next_node = next_node

  def get_value(self):
    return self.value

  def get_next_node(self):
    return self.next_node

  def set_next_node(self, next_node):
    self.next_node = next_node

# Our LinkedList class
class LinkedList:
  def __init__(self, value=None):
    self.head_node = Node(value)

  def get_head_node(self):
    return self.head_node

  def insert_beginning(self, new_value):
    new_node = Node(new_value)
    new_node.set_next_node(self.head_node)
    self.head_node = new_node

  def stringify_list(self):
    string_list = ""
    current_node = self.get_head_node()
    while current_node:
      if current_node.get_value() != None:
        string_list += str(current_node.get_value()) + "\n"
      current_node = current_node.get_next_node()
    return string_list

  # Define your remove_node method below:
  def remove_node(self, value_to_remove):
    current_node = self.get_head_node()
    if current_node.get_value() == value_to_remove:
      self.head_node = current_node.get_next_node()
    else:
```

```
    while current_node:
      next_node = current_node.get_next_node()
      if next_node.get_value() == value_to_remove:
        current_node.set_next_node(next_node.get_next_node())
        current_node = None
      else:
        current_node = next_node
```